Computer Science Department

TECHNICAL REPORT

Relational Database Behavior: Utilizing Relational Discrete Event Systems and Models

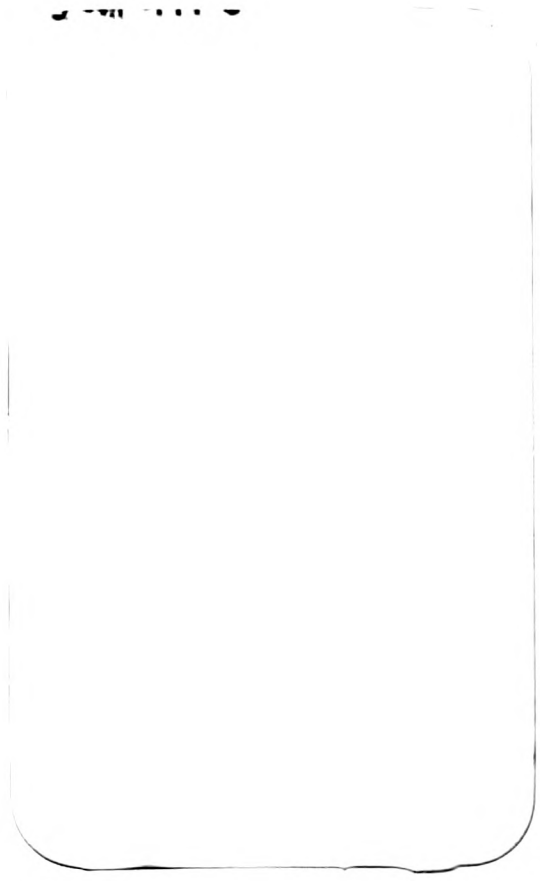
Z. M. Kedem
*A. Tuzhilin**

Technical Report 404

October 1988

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



Relational Database Behavior: Utilizing Relational Discrete Event Systems and Models

Z. M. Kedem
*A. Tuzhilin**

Technical Report 404

October 1988

*This work was partially supported by the Office of Naval Research under the contract N00014-85-K-0046

Abstract

Behavior of relational databases is studied within the framework of *Relational Discrete Event Systems* (RDES) and *Models* (RDEM). Production system and recurrence equation RDEMs were introduced and their expressive powers compared. Non-deterministic behavior was defined for both RDEMs and the expressive power of deterministic and non-deterministic production rule programs was also compared. Both comparisons show that non-determinism increases expressive power of production systems. A formal concept of a production system interpreter was defined, and several specific interpreters were proposed. One interpreter, called *parallel deterministic*, was shown to be better than others, including the conflict resolution module of OPS5, in many respects.

1 Introduction

There has been much work done on studying behavioral aspects of databases. Triggers [1,11] and alerters [4], database models that explicitly support behavior, like RUBIS [8], SHM+ [2], Event Model [6], some recent attempts to interface production systems and databases [5,10] – are examples of this research.

However, there is no formal framework for studying behavior which allows to compare the proposed models in terms of their behavioral properties as relational query languages can be compared in terms of expressive power and relational completeness. Because of that, there are many different approaches for specifying behavior which yet have to be brought together in a single formal framework.

This report contributes towards the development of theoretical models of database behavior by introducing the concepts of *Relational Discrete Event System (RDES)* and *Relational Discrete Event Model (RDEM).* RDES is defined as a set of trajectories over the space of consistent relational database states and is based on the notion of a Discrete Event System (DES) [14]. RDEM is a formalism, based on the relational data model, that describes a possibly *infinite* RDES set in *finite* terms. These two concepts are important because they allow to compare any two behavioral models based on the relational data model in terms of the RDESes their RDEMs describe.

As the first step in our research, we consider the *production system RDEM (PS RDEM)* and the *recurrence equation RDEM (RE RDEM)* and compare them in terms of the RDESes they describe. Both of these RDEMs are *non-deterministic.* Non-determinism is important for describing systems behavior for two reasons. First, many real-life systems exhibit non-deterministic behavior. Second, as we will show, it adds expressive power to production systems. Some aspects of non-determinism were previously studied in the framework of Datalog [7] but not in the context of behavior of relational databases.

We also define formally the concept of an interpreter for the PS RDEM and consider several specific interpreters. Formal comparison of interpreters constitutes one of the main contributions of the report. A particular interpreter, called *parallel deterministic*, was proven to be better than others in many respects. As a special case, this result shows that the OPS5 interpreter [1] is inferior to the parallel deterministic interpreter in a formal sense.

[1] We associate the meaning with the term "OPS5 interpreter" that is *different* from the one used in the

2 Relational Discrete Event Systems and Models

Let **R** be a database schema with *finite* domains. Let $\Sigma(\mathbf{R})$ be the set of all *consistent* database states with the schema **R**. Note that $\Sigma(\mathbf{R})$ is a finite set. Then a *Relational Discrete Event System (RDES)* is a set of finite and infinite sequences, called *trajectories*, over the database states from $\Sigma(\mathbf{R})$. In other words, RDES is a Discrete Event System [14,9], where the state space is $\Sigma(\mathbf{R})$. Intuitively, an RDES defines the class of all possible behaviors of a relational database, and a trajectory represents an element of this class, i.e. one possible evolution of the database in time. The set of all trajectories, RDES, is infinite in general. A *Relational Discrete Event Model (RDEM)* is a *finite* mathematical description of the possibly *infinite* set RDES. In other words, RDEM is a Discrete Event Model [13] for an RDES.

Two RDEMs will be considered in this paper. Section 3 describes an RDEM based on production systems and section 5 describes an RDEM based on recurrence equations.

3 Production System RDEM

A *Production System Relational Discrete Event Model (PS RDEM)* is a production system (D, P, I) [3], where the working memory D is a point in $\Sigma(\mathbf{R})$, i.e. is a consistent instance of the relational schema **R**, P is a set of production rules based on the underlying relational model, and I is an interpreter. We define production rules in the next paragraph. Properties of various interpreters I constitute one of the central issues of the report and will be studied later.

Production rules are defined as follows. Let P_i be a literal, i.e. a predicate from **R**, a relational operator ($=, \geq$, etc.) or their negation. Let O_{ij} be INSERT, DELETE or UPDATE operator. The format of the INSERT or DELETE operator is $< op > (R, x_1, x_2, \ldots, x_m)$, where $< op >$ is either INSERT or DELETE, x_i are *all distinct* variables, and R is a relation to be modified. UPDATE is defined as UPDATE$(R, x_1, \ldots, x_m; x_1 = f_1(x_1, \ldots, x_m); \ldots; x_m = f_m(x_1, \ldots, x_m))$, where f_i is a function from $dom(R.A_1) \times \ldots \times dom(R.A_m)$ to $dom(R.A_i)$ and x_j is defined on the attribute $R.A_j$. $\otimes$ denotes the EXCLUSIVE-OR op-

OPS5 literature [3]. In [3] the rules interpreter is the part of the inference engine that executes the rules. In this report, "OPS5 interpreter" is the collection of selection strategies (either LEX or MEA) used for the conflict resolution.

erator, i.e. $O_1 \otimes O_2$ means: either apply operator O_1 or operator O_2 but not both. Then the format of a production rule is

$$\bigwedge_{i=1}^{m} P_i(x_{i1}, \ldots, x_{ik_i}) \rightarrow \otimes_{j=1}^{n} O_{j1}; O_{j2}; \ldots; O_{jl_j}$$

The operator $\otimes$ expresses non-determinism among operations, e.g. "if a person enters a bank, he/she can go to the teller's line or to the automatic tellers machines." We do not study this kind of non-determinism in this report, i.e. we always assume that $n = 1$. Only *safe* rules, in the sense of [12, pp. 104–106], will be considered in this report. This implies, among other things, that any variable in any operator O_{ij} must also appear in some predicate P_i. The variables in a rule will be either *deterministic* or *non-deterministic*. This important concept is explained by the following example.

Example. Consider a simplified airline system AIRLINE. An airline has a number of planes that travel between airports. Once a plane arrives at an airport, it is scheduled for landing. Once it lands, it moves to a terminal and then to the takeoff queue. After the takeoff, a plane flies to another airport. This process continues indefinitely.

A state of AIRLINE is defined by the following relations. $AIRPORT(A, TRM)$: an airport A has a terminal TRM; $DOCK(A, TRM, P)$: a plane P is docked at a terminal TRM in an airport A; $LQ(A, P, POS)$: a plane P is in position POS in the landing queue of an airport A; $TQ(A, P, POS)$: a plane P is in position POS in the takeoff queue of an airport A; $DEST(P, A)$: a plane P is flying towards an airport A. The function $NEXT(P, A)$ specifies which airport a plane P should fly to after an airport A.

Two examples of rules are provided below.

P1: If an airplane is the first in the landing queue and there are free terminals then move the plane *non-deterministically* to one of the free terminals.
$LQ(A, P, POS) \wedge POS = 1 \wedge AIRPORT(A, TRM) \wedge (\forall P')\neg DOCK(A, TRM, P')$
$\rightarrow DELETE(LQ, A, P, 1);\ INSERT(DOCK, A, TRM, P)$

P2: If an airplane is the first in the take-off queue then it will take off and fly to its next destination.
$TQ(A, P, POS) \wedge POS = 1 \wedge A' = NEXT(P, A) \rightarrow$
$DELETE(TQ, A, P, POS); INSERT(DEST, P, A')$

As part of the definition of AIRLINE, the variable TRM in **P1** is specified to be *non-deterministic*. In this case, a choice is made among all the available terminals in the airport, and only one terminal is selected for docking. The specification of which variables are non-deterministic is a part of the formal syntax; however, we have not done so in this report to simplify the exposition.

A set of production rules will be called a **Prodata program**. It is an acronym stressing synergy of PROduction rules and DATAbases since the model fully supports both relational database and production system models. A Prodata program is *rule deterministic* if all the variables in all its rules are deterministic. Otherwise, it is called *rule non-deterministic*.

We impose two additional *consistency restrictions* on the set of Prodata programs. First, if there are two rules $P_1 \rightarrow UPDATE(R, \ldots . A, \ldots ; A = F_1(\ldots .) \ldots)$ and $P_2 \rightarrow UPDATE(R, \ldots . A \ldots . ; A = F_2(\ldots .) \ldots)$ then either $F_1 = F_2$ or $P_1 \wedge P_2 = FALSE$, thus precluding conflicts between updates. Second, if there are two rules $P_1 \rightarrow INSERT(R, x_1, x_2, \ldots , x_k)$ and $P_2 \rightarrow DELETE(R, x_1, x_2, \ldots . x_k)$ then $P_1 \wedge P_2 = FALSE$, thus precluding conflicts between inserts and deletes.

Interpreter. An interpreter is the third component of a production system. It will be described together with the recognize-act cycle of a production system. The *recognize-act cycle* consists of the following steps. First, each rule is matched against the current state of the database. This results in a set of *instantiated tuples*, which is called a *conflict set*.

Second, non-deterministic choices are made in the conflict set. For each rule, fix the deterministic variables and select only one set of values of the non-deterministic variables from all the sets of values that render the left hand side (LHS) of the rule true. The resulting reduced conflict set is converted into an *operation set* by creating operations out of each instantiated tuple from the conflict set.

Third, a subset of operations is selected for the execution. An *interpreter* does this selection. Specifically, for a set of operations O, an interpreter I defines a set $\mathbf{S}(O) = \{O_1, O_2, \ldots , O_n\}$ of subsets of O ($O_i \subset O$). The interpreter defines $\mathbf{S}(O)$ independently of the recognize-act cycles, and, therefore, $\mathbf{S}(O)$ depends only on O and is the same for any cycle. After the interpreter selects the set $\mathbf{S}(O)$, all operations in the set $O_i \in \mathbf{S}(O)$ are executed simultaneously. This is done non-deterministically for all $i = 1, \ldots , n$, which means that n new states are generated after the execution. Therefore, a trajectory splits

into n non-deterministic trajectories. This completes the recognize-act cycle.

If always $n = 1$ in the definition of the set $\mathbf{S}(O)$ then the interpreter is called *operationally deterministic* to distinguish this kind of determinism from the rule determinism. Otherwise, the interpreter is called *operationally non-deterministic.* We consider the following four kinds of interpreters. A *universal* interpreter is the one that non-deterministically selects *any* subset of O for execution. If $n = 1$ and $O_1 = O$ then such an interpreter is called *parallel operationally deterministic.* If $n = |O|, O_i = \{x_i\}$ where $x_i \in O$ then such an interpreter is called *selective operationally non-deterministic.* If there is a function such that for any operation set it selects one element out of that set then the corresponding interpreter is called *selective operationally deterministic.*

A *trajectory* $TR_I(D, P)$, generated by the production system (D, P, I), is an infinite sequence $D_0, D_1, \ldots$ or a finite sequence $D_0, D_1, \ldots, D_n$ of states such that $D_0 = D$ and D_{i+1} belongs to the set of states non-deterministically obtained from D_i. If no rules match D_k then the sequence is finite. A *RDES generated by PS RDEM* (D, P, I) is the set of all the trajectories $TR_I(D, P)$.

We described a production system which is a modification of OPS5. Specifically, the following changes to OPS5 have been made. First, we removed restrictions of OPS5 we feel are unnecessarily limiting such as the requirement that attributes of a condition element must belong to the same element class (relation schema), and the requirement that the first condition element cannot be negated. Second, the structure of production rules is made more relational and in line with logic programming. Third, only conceptually relevant features of OPS5 were left to simplify the model. For example, our model does not contain OPS5 actions such as *bind, build, write, accept, call.* Fourth, non-determinism was considered. Finally, as will be shown in the next section, the OPS5 interpreter is inferior to some of our interpreters in a certain formal sense.

Our model also differs from the RPL model [5]. First, RPL model is based on SQL (the LHS of a rule is SQL SELECT and the RHS is INSERT, DELETE or UPDATE statements), and our model borrowed many features from logic programming. Second, conflict sets are computed differently in the two models. Third, we also consider non-determinism and various non-deterministic interpreters.

4 Comparison of Various Interpreters

In this section we study relationships among various interpreters. Specifically, we define different kinds of "dominations" of one interpreter over the other. Then we compare the interpreters, defined in the previous section, in terms of these dominances. This allows us to see which interpreters have more expressive power and, therefore, are suited better for the description of RDESes.

Definition 1 An interpreter I *absolutely dominates* an interpreter I' if for any relational schema $\mathbf{R}$, any Prodata program P, there is a Prodata program P' such that for any database instance R with schema $\mathbf{R}$, RDES generated by (R, P', I') is the same as the RDES generated by (R, P, I).

Definition 2 An interpreter I *strongly dominates* an interpreter I' if for any relational schema $\mathbf{R}$, any instance R of that schema and any Prodata program P, there is a Prodata program P' such that RDES generated by (R, P', I') is the same as the RDES generated by (R, P, I).

Definition 3 The interpreter I_1 *(deterministically) path-dominates* the interpreter I_2 if for any relational schema $\mathbf{R}$, any instance R of that schema, any Prodata program P_2 and any non-deterministic trajectory $TR_{I_2}(R, P_2)$, there exists a *(deterministic)* program P_1 and a trajectory $TR_{I_1}(R, P_1)$ such that the two trajectories coincide. In other words, the set of all possible trajectories generated by I_2 is contained in the set of all the trajectories generated by I_1. This kind of dominance will also be called *path-dominance.*

Two interpreters are *equivalent* if they dominate each other. This definition is applicable to all four types of dominances.

Note that $TR_{I_1}(R, P_1)$ in the definition of deterministic path-dominance is unique for *deterministic* interpreters. Also, note that all four types of dominances constitute partial orderings on the class of interpreters.

It is easy to see that absolute dominance implies strong dominance, and strong dominance implies path-dominance. Also, deterministic path-dominance implies path-dominance but deterministic path-dominance is unrelated to strong and absolute dominance.

The following theorem says that rule non-determinism adds expressive power to production systems.

Theorem 1 *For any interpreter, there is a rule non-deterministic production system such that its RDES cannot be generated by any rule deterministic production system under any deterministic interpreter.*

Properties of parallel deterministic interpreter are studied now.

Theorem 2 *The parallel deterministic interpreter absolutely dominates any other interpreter, including the universal one.*

The theorem holds because all the domains are finite. The parallel deterministic interpreter dominates the universal one because operational non-determinism can be "converted" into rule non-determinism for finite domains.

Proposition 1 *The universal interpreter path-dominates any other interpreter.*

It follows from Theorem 2 and Proposition 1 that the universal and parallel deterministic interpreters are equivalent in terms of the path-dominance.

However, the previous result does not hold for deterministic path dominance:

Proposition 2 *Parallel deterministic interpreter does not path-dominate the universal one deterministically.*

We can establish weaker results about deterministic path-dominance of the parallel deterministic interpreter. First, we provide some preliminary definitions.

Definition 4 The interpreter I_1 *path-dominates (deterministically)* the interpreter I_2 *on a set of trajectories* $\mathbf{T}$, if for any schema $\mathbf{R}$, any initial state of that schema D_0, any Prodata program P_2 and any non-deterministic trajectory $TR_{I_2}(D_0, P_2)$ *from* $\mathbf{T}$, there exists a (deterministic) program P_1 and a trajectory $TR_{I_1}(D_0, P_1)$ such that this trajectory belongs to $\mathbf{T}$ and the two trajectories coincide.

Let $\mathbf{R}$ be the set of states and $\alpha, \beta \in \mathbf{R}^*$. Denote $\beta\beta\ldots\beta\ldots$ as (β) and call it a *cycle*. A *cyclic* trajectory is a trajectory of the form $\alpha(\beta)$, i.e. it has an initial string followed by a cycle. Cyclic trajectories constitute an important class of trajectories because they define behavior of rule deterministic systems under operationally deterministic interpreters.

Proposition 3 *Parallel deterministic interpreter deterministically path-dominates the universal one on the set of cyclic trajectories.*

Proposition 4 *Parallel deterministic interpreter deterministically path-dominates any operationally deterministic interpreter on the set of rule deterministic programs.*

The next two results show, among other things, that the OPS5 interpreter is not the best one in terms of the dominance criterion.

Proposition 5 *Parallel deterministic interpreter strictly path-dominates the selective non-deterministic one.*

Proposition 6 *Selective non-deterministic and selective deterministic interpreters are equivalent in terms of path-dominance.*

Corollary. *The parallel deterministic interpreter strictly deterministically path-dominates the selective deterministic interpreter, including the OPS5 interpreter.*
(Follows from Propositions 4, 5 and 6.)

Implications of this section's results are discussed in the conclusion.

5 Recurrence Equation RDEM

Let $\mathbf{R} = (R_1(A_{11}, \ldots, A_{1k_1}), \ldots, R_m(A_{m1}, \ldots, A_{mk_m}))$ be a database schema and D be defined as before, i.e. $D \in \Sigma(\mathbf{R})$. The behavior is described as follows.

Let $D^{(n)} = (R_1^{(n)}, R_2^{(n)}, \ldots, R_m^{(n)})$ be the state of the database at time n. Then the recurrence equation RDEM is defined as $D^{(n+1)} = \mathbf{F}(D^{(n)})$, where $\mathbf{F}$ is a vector valued function defined by

$$R_i^{(n+1)}(x_{i1}, \ldots, x_{ik_i}) = \begin{cases} TRUE & \text{if } P_i(x_{i1}, \ldots, x_{ik_i}) \\ FALSE & \text{if } Q_i(x_{i1}, \ldots, x_{ik_i}) \\ R_i^{(n)}(x_{i1}, \ldots, x_{ik_i}) & \text{otherwise} \end{cases}$$

$$R_i^{(n+1)}.x_{ij} = \begin{cases} f_{ij1}(x_{i1}, \ldots, x_{ik_i}) & \text{if} P_{i1}(x_{i1}, \ldots, x_{ik_i}) \\ f_{ij2}(x_{i1}, \ldots, x_{ik_i}) & \text{if} P_{i2}(x_{i1}, \ldots, x_{ik_i}) \\ \ldots & \\ f_{ijm}(x_{i1}, \ldots, x_{ik_i}) & \text{if} P_{im}(x_{i1}, \ldots, x_{ik_i}) \\ R_i^{(n)}.x_{ij} & \text{otherwise} \end{cases}$$

where P_i, Q_i, P_{ij} are disjunctive normal form (DNF) formulas with predicates $R_1, \ldots, R_m$, relational operators ($=, \geq, etc.$) and their negations, such that $P_i \wedge Q_i = FALSE$, $P_{ij} \wedge P_{il} = FALSE$ for $j \neq l$. Moreover, conjunctive clauses in these DNF formulas are *safe* [12, pp. 105]. f_{ijl} in the second formula is a function from $dom(R_i.A_{i1}) \times \ldots \times dom(R.A_{im})$ to $dom(R.A_j)$. The variables x_{ij} are either *deterministic* or *non-deterministic* (see Section 3).

RE RDEM has the following semantics. For each predicate $P_{ij}(x_{i1}, \ldots, x_{ik_i})$ and $P_i(x_{i1}, \ldots, x_{ik_i})$ fix deterministic variables. Select only one set of values of non-deterministic variables from all the sets of values that render P_{ij} true. Now, both deterministic and non-deterministic variables have values assigned to them. For this assignment, compute $D^{(n+1)}$. One choice among non-deterministic variables yields one trajectory. The set of all choices yields the set of trajectories which will be called trajectories *generated* by a given RE RDEM.

The first equation corresponds to the INSERT/DELETE operations, i.e. if a tuple is inserted into the relation R_i then $R_i^{(n+1)}(x_{i1}, \ldots, x_{ik_i}) = TRUE$, and if it is deleted then the value is $FALSE$. The second equation corresponds to the UPDATE operation. It indicates how attributes are changed after the update.

Example. Consider the AIRLINE system again. The recurrence equation for TQ is:

$$TQ(A,P,POS) = \begin{cases} TRUE & \text{if } DOCK(A,TRM,P) \wedge POS = SIZE(TQ)+1 \\ FALSE & \text{if } TQ(A,P,POS) \wedge POS = 1 \\ TQ(A,P,POS) & \text{otherwise} \end{cases}$$

This equation has only deterministic variables.

The following theorems show that PS RDEM has more expressive power than RE RDEM but deterministic PS and RE RDEMs have the same expressive power. In these theorems we assume that the interpreter for PS RDEM is the parallel deterministic one.

Theorem 3 *For any system of recurrence equations there is a Prodata program such that the two models generate the same RDES.*

Theorem 4 *There is a (non-deterministic) PS RDEM such that no RE RDEM can generate the same RDES for it.*

Theorem 5 *For any deterministic Prodata program there is a system of recurrence equations such that the two models generate the same RDES.*

6 Conclusions and Future Work

In this report, we introduced a unifying framework for studying database behavior based on the concepts of *relational discrete event system* and *relational discrete event model*. Also, production system and recurrence equation RDEMs were presented and compared in terms of their expressive power. In addition, non-deterministic behavior of databases was defined for both RDEMs, and the expressive power of deterministic and non-deterministic Prodata (sets of production rules) programs was compared. It was shown that the recurrence equation RDEM has less expressive power than the production system RDEM for non-deterministic Prodata programs. However, for deterministic programs, they have the same expressive power. For Prodata programs, it was shown that non-deterministic programs have more expressive power than deterministic ones for deterministic interpreters. This exhibits importance of non-determinism for describing certain types of database behavior.

For PS RDEM, several interpreters were proposed and compared in the framework of various types of dominance. Among these interpreters, parallel deterministic interpreter is the most promising. It was shown that it absolutely dominates any other interpreter, although it may not path-dominate all of them deterministically. Nevertheless, for the important class of cyclic trajectories, parallel deterministic interpreter deterministically path-dominates any other interpreter. In addition, it deterministically path-dominates any operationally deterministic interpreter on the set of rule deterministic programs. It was also shown that the parallel deterministic interpreter strictly deterministically path-dominates an OPS5-like interpreter. This means that OPS5 is not the best interpreter, compared in terms of the expressive power.

We are currently working on the ways to reduce the number of non-deterministic choices in database behavior by preferring some choices to the others. For example, in the chess game, a player makes only those moves that improve his/her position, as opposed to all possible moves the player can make.

References

[1] M. Astrahan and et al. System R: a relational approach to data. *TODS*, 97–137, June 1976.

[2] M. L. Brodie and D. Ridjanovic. On the design and specification of database transactions. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling*, chapter 10, Springer-Verlag, 1984.

[3] L. Brownston, R. Farrell, and E. Kant. *Programming Expert Systems in OPS5: an Introduction to Rule-Based Programming.* Addison-Wesley, 1986.

[4] P. Bunemann and E. Clemons. Efficiently monitoring relational data bases. *TODS*, September 1979.

[5] L. M. L. Delcambre and J. N. Etheredge. A self-controlling interpreter for the relational production system. In *Proceedings of ACM SIGMOD Conference*, pages 396–403, 1988.

[6] R. King and D. McLeod. A unified model and methodology for conceptual database design. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling*, pages 313–327, Springer-Verlag, 1984.

[7] R. Krishnamurthy. Non-deterministic choice in Datalog. In C. Beeri, editor, *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, 1988.

[8] J. Y. Lingat, P. Nobecourt, and C. Rolland. Behavior management in database applications. In *International Conference on Very Large Databases*, pages 185–196, 1987.

[9] P. Ramadge. Supervisory control of discrete event systems: a survey and some new results. In *Lecture Notes in Control and Information Sciences, 103*, pages 69–80, Springer-Verlag, 1987.

[10] T. Sellis, C. Lin, and L. Raschid. Implementing large production systems in a DBMS environment: concepts and algorithms. In *Proceedings of ACM SIGMOD Conference*, pages 404–412, 1988.

[11] M. Stonebraker. Triggers and inference in data base systems. In M. Brodie and J. Mylopoulos, editors, *On Knowledge Base Management Systems*, pages 297–314, Springer-Verlag, 1986.

[12] J. Ullman. *Principles of Database and Knowledge-Base Systems*. Volume 1. Computer Science Press, 1988.

[13] P. Varaiya. Preface to the proceedings. In *Discrete Event Systems: Models and Applications*, Springer-Verlag, 1987. Lecture Notes in Control and Information Sciences, 103.

[14] P. Varaiya and A. Kurzhanski, editors. *Discrete Event Systems: Models and Applications*, Springer-Verlag, 1987. Lecture Notes in Control and Information Sciences, 103.

NYU COMPSCI TR-404
Kedem, Zvi M
Relational database
behavior c.2